An Experimental Study on Fitness Distributions of Tree Shapes in GP with One-Point Crossover

César Estébanez, Ricardo Aler, José M. Valls, and Pablo Alonso

Universidad Carlos III de Madrid
Avda. de la Universidad, 30, 28911, Leganés (Madrid). Spain
{cesteban,aler,jvalls}@inf.uc3m.es, pablo.alopez@alumnos.uc3m.es

Abstract. In Genetic Programming (GP), One-Point Crossover is an alternative to the destructive properties and poor performance of Standard Crossover. One-Point Crossover acts in two phases, first making the population converge to a common tree shape, then looking for the best individual within that shape. So, we understand that One-Point Crossover is making an implicit evolution of tree shapes. We want to know if making this evolution explicit could lead to any improvement in the search power of GP. But we first need to define how this evolution could be performed. In this work we made an exhaustive study of fitness distributions of tree shapes for 6 different GP problems. We were able to identify common properties on distributions, and we propose a method to explicitly evaluate tree shapes. Based on this method, in the future, we want to implement a new genetic operator and a novel representation system for GP.

1 Background and Motivation

Genetic Programming (GP) is a very successful stochastic search technique, widely used by researchers to solve complex problems. Thirty six human competitive results have been documented in the last years, and there are twenty three instances where GP has duplicated the functionality of a previously patented invention, infringed a previously issued patent, or created a patentable new invention [1,2].

However there are many controversial issues related to classical GP. Among the most important ones is the destructive effects of Standard Crossover, which have been widely documented in GP literature [3,4,5,6]. Another major issue is the phenomena known as Code Bloat [3,5,6,7,8,9]. Bloat appears when individuals in the GP population tends to rapidly grow over the generations, producing very large and slow trees. It is a very serious problem because it consumes a lot of resources without a proportional improvement of the average fitness. In [10], Langdon and Poli explain some possible theories of why Code Bloat occurs, and it happens that most of them suggest that bloat is a consequence of destructive effects of Standard Crossover.

In [11] Standard Crossover is compared with two new crossover operators, namely: One-Point Crossover and Uniform Crossover. They conclude that Standard Crossover is a biased and local search operator, which cannot explore the

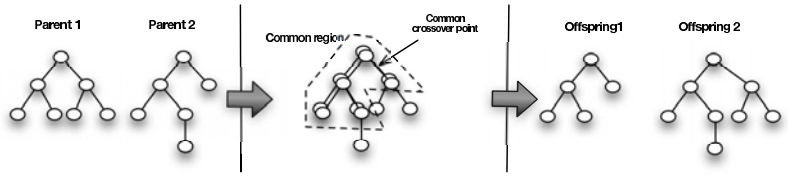


Fig. 1. One-Point Crossover operator

search space quickly, that can only reach certain areas of the search space, and that can get stuck in local maxima. Some documented results show that Standard Crossover has a limited performance advantage compared to mutation based operators [12,13]. On the other hand, One-Point Crossover shows some interesting properties. It works this way: When the two parents are selected, they are aligned. Both trees are traversed to identify the parts that have the same shape. All the links in the common region are stored. The traverse is stopped in a branch as soon as an arity mismatch between corresponding nodes in the two trees is present. Finally, a crossover point is randomly selected among the stored links, and the resulting subtrees are swapped. Figure 1 shows an example of One-Point Crossover.

One-Point Crossover has some interesting features, as we already mentioned. First, it is immune to Bloat [10]. Second, if the initial population is correctly generated, then the individuals of the earlier generations should have very small common regions. This means that crossover points will probably be selected very near to the root of the parents, while in standard crossover every link of the tree has same probability. The consequence is that the amount of genetic material exchanged is greater when using one-point crossover. After a number of generations, (in absence of mutation) one-point crossover makes the population evolve to a common tree shape. From this moment, GP behaves like a Genetic Algorithm (GA), selecting common crossover points in fixed-length-and-shape GP individuals. In this phase, GP intensifies exploitation of the region of the search space bounded by the selected tree shape.

1.1 Evolution of Tree Shapes

From the point of view of this work the most important fact about One-Point Crossover is that, in absence of mutation, it makes the population to converge to a common tree shape. So, One-Point Crossover uses the first generations of the GP run for select one tree shape, and then it makes the whole population to adopt that shape. Based on this fact, we claim that One-Point Crossover is performing an implicit evolution of tree shapes. This makes perfect sense for us: A tree shape can be seen as a region of the search space, namely the region containing all the solutions which are represented by a tree with that shape. Using this point of view, one can imagine One-Point Crossover performing a search in two phases: First, it makes a global search of tree shapes, trying to

identify promising regions of the search space; Then, it finally focusses on a specific region, considered to be interesting, and it looks for the best solution inside that region.

If this point of view is correct, One-Point Crossover must be following a criterion to prefer some tree shapes to others. But it has not explicit information on tree shapes. The only information available during the run is the fitness of the GP individuals on the population. Being the fitness function the only information available, it seems to be clear that tree shapes are selected using the fitness of the GP individuals that have that shape, and by the fitness of those individuals containing the building blocks needed to construct the shape[1]. It is very important to remark that those fitness measures are considered one by one, instead of looking at the global picture of all the available matches of the considered tree shape.

We believe that this could be not the best way of evaluate a shape. First, because it is unfair: depending on how the initial population samples the search space of all the possible tree shapes, some shapes will have more matches in the population than others. That means more probabilities of sampling an better-than-average-fitness match, and therefore more opportunities to proliferate. Second, because this method can be deceptive: Big tree shapes can contain billions of different matches. If we have an better-than-average-fitness match in the population, the shape of that match will proliferate, even if the population also contains other matches of that shape with very bad fitnesses. Out intuition says that it will be much more accurate to have a general look on all the available matches and evaluate the shape using a joint fitness measure.

The objective of this work is to find an explicit method to evaluate tree shapes, and for that, we made an experimental study on the fitness distributions of GP tree shapes. First, we need to define what it is a good or a bad tree shape, then, we want to know how to distinguish between them. Finally, we want to obtain a fitness function that can explicitly evaluate a tree shape, that has a reasonable computational complexity, and that can be reused on a great variety of different GP problems. If we can find such a function, then we could improve the search capabilities of One-Point Crossover, making the convergence criterion become clear, fair, and rigorous. In the next section we explain the methodology we followed to study the fitness distributions, and we present the exhaustive experiments we carried out. Then, in Section 3 we define the fitness criterion we will follow to measure tree shapes. We formulate that definition using the data extracted from the experiments. In Section 4 we show how computationally-hungry can be the explicit fitness function we proposed. Finally in Section 5, we explain the conclusions we obtained from this work, and we discuss some

[1] Depending on the selection methods and the genetic operators used, the propagation rate of tree shapes from one generation to the next will vary. The GP Schema Theorems proposed by Poli and Langdon in [10] describe the dynamics of tree shapes on GP + One-Point Crossover. In this work it is enough to consider that tree shape can appear in a population because of a verbatim copy (reproduction, elitism, etc.), or because of the combination of building blocks performed by One-Point Crossover.

remaining issues about the future possibilities of this new, explicit fitness function for tree shapes.

2 Fitness Distributions of Matches

The goal of this research is to answer three questions:

- How can we tell whether a tree shape is good or bad?
- Can we estimate that information in a general, accurate, rigorous way?
- Can we make this estimation efficiently?

In order to give answers, we coded six different GP problems in ProGen[2]. Two of them are classical GP problems of two different classes: Symbolic Regression and Boolean 6-Multiplexer. The third problem is a real application of Genetic Programming called GP-Hash [14]. The last three problems are applications of GP to classification, using three UCI datasets: Pima Indians Diabetes [15], SpectHeart [16], and Haberman' Survival [17]. In Table 1 we briefly show the configuration of the six GP problems.

Table 1. Parameters of the 6 GP problems

	Functions	**Terminals**	**Fitness (minimize)**	**Init Method**
Regression	+,−, ∗, /	X	Prediction Error	Grow, depth 3-6
6-Multiplexer	AND, OR, NOT, IF	A0, A1, D0, D1, D2, D3, D4	Prediction Error	Grow, depth 3-9
GP-Hash	>>, AND, OR, NOT, XOR, MULT, SUM	a0, hval	Avalanche Effect	Grow, depth 2-6
Diabetes	+, −, ∗, %, ∗ − 1, <, >, =, AND, OR, NOT, IF-ELSE	Attributes of the dataset	Classification Error	Grow, depth 3-5
Haberman	+, −, ∗, %, ∗ − 1, <, >, =, AND, OR, NOT, IF-ELSE	Attributes of the dataset	Classification Error	Grow, depth 3-5
SPECT Heart	+, −, ∗, %, ∗ − 1, <, >, =, AND, OR, NOT, IF-ELSE	Attributes of the dataset	Classification Error	Grow, depth 3-5

For each problem, we randomly generated one hundred tree shapes. Shapes were internally represented in ProGen as GP schemata of order 0 (GP trees with don't-care symbols "=" in all their nodes, see [10] and Figure 3). For each shape, we exhaustively generated all its matches and measured their fitness. That allows us to construct the complete fitness distribution of each tree shape for each problem. We show some of those distributions in Figure 2 for illustrative purposes. It

[2] ProGen is an Open Source GP framework designed at Universidad Carlos III de Madrid and entirely coded in Java.

took us around one month of intense computation in a 8-Core Intel Xeon with 8Gb of RAM to produce the output files (more than 13Gb of data), and we could only explore relatively small tree shapes. In last column of Table 1 we show the initialization method used in each problem (size of distribution highly depends not only on the tree shape size but also on the size of the function set, so the shape size have to be readjusted for each problem). We had to keep the initial depth at small levels, and we always used grow method. In the future we want to invest more resources to repeat these experiments on bigger shapes (currently, we are working on a clusterized version of ProGen that should drastically reduce the execution times). Anyway, even with relatively small initial depths, most of the tree shapes we evaluated had millions of matches. Furthermore, according to Poli and Langdon [4], the proportion of individuals of a given fitness is generally independent of program size, so we expect a similar behavior on bigger shapes.

Studying in depth the distributions, we can conclude:

- Fitnesses of matches do not follow a normal distribution or any other known distribution (we used Shapiro-Wilkis tests on those problems with gaussian-look distributions).
- The distributions are highly problem dependent.
- Almost all the studied distributions are extremely leptokurtic, with a very large proportion of matches grouping very close to the mode.
- The mode usually coincides with the worst possible fitness.

All those conclusions were expected: As each tree shape encodes a large number of possible solutions, it is reasonable that most of them lack of any sense (worst possible fitness), and only a very few are good solutions that have a decent fitness. We also expected the problem dependency: Obviously, the way fitnesses distribute depends a lot on the definition of the fitness function. We can identify two different types of fitness functions in our problems: First, we have two fitness functions (Regression and GP-Hash) which have no upper bound (it can get as big as infinite). Those distributions have huge tails that we filtered before processing them. The other fitness functions do have an upper bound, and the worst fitness is not that bound, but the middle point between that point and 0: In a binary classification problem, having an error of 100% is as hard as having 0% (just invert the output and you have a perfect classifier).

3 A Fitness Function for Tree Shapes

Fitness distributions of shapes do not follow any known distribution, so we have to find any other statistical information for evaluate tree shapes. We extract a set of statistics from the distribution of each tree shape of each of the 6 GP problems: Minimum, first quartile, median, mean, third quartile, and maximum. From our point of view, the most interesting one is the first quartile: Minimum value is interesting, because it represents the best solution you can find inside that tree shape. But tree shapes are regions of the search space that conglomerate millions of possible solutions. An outstanding minimum lost in a huge population

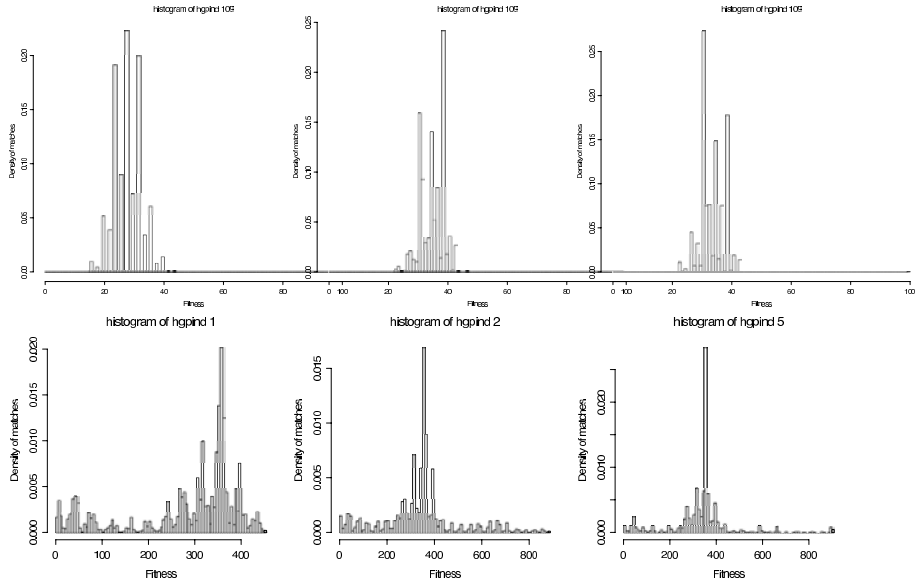
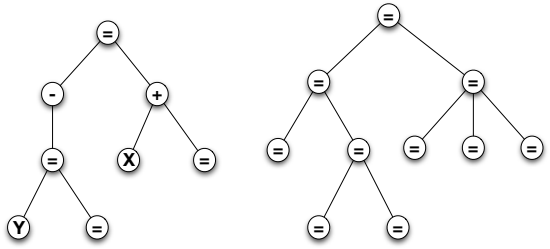


Fig. 2. Some illustrative histograms of fitness distributions of two different problems. First row: Boolean 6-Multiplexer; Second row: Symbolic Regression.



(a) Order 4 schema (b) Order 0 schema

Fig. 3. Two GP schemata of different orders

of billions of bad solutions is not helpful at all. After finding a good shape, we want to be able to find the good solutions it contains, and that will be much easier in a region with a large proportion of good solutions. This proportion could be measured by the first quartile. A low first quartile guarantees that you will find solutions as good as or better than it with a probability of 25%.

The problem with first quartile is that it is not easy to calculate from small samples, so it could lose its utility when we cannot use complete distributions any more. Mean is very easy to estimate, even with unknown underlying distributions. So, for each GP problem, we generated a table with a row for each tree shape, and a column for each statistic, and we sorted the rows by the mean column. Table 2 shows the head and the tail of the table for the Regression

Table 2. Head and tail of the statistics table of the GP-Hash problem. Rows are ordered by mean.

	Min	FirstQu	Median	Mean	ThirdQu	Max
23	0	52,87	217	176	279,1	318,4
85	0	93,91	274,2	223,1	322,6	353,4
64	5,339	109	279,1	230,9	317,7	346,5
79	0	190,6	288,9	244,7	331,8	353,4
3	0	133	298,2	246,2	351,5	366,3
47	0	155,9	307,9	247,6	346,5	366,3
...	...	...	...	...	...	...
98	0	269,4	351,5	348,4	419,1	804,4
27	0	307,9	356,4	349,5	366,3	842,2
83	0	279,1	351,5	350,7	366,3	1063
1	0	279,1	351,5	351,5	400,2	822,7
31	0	307,9	356,4	353,2	366,3	982
13	0	279,1	356,4	355	385,3	1063

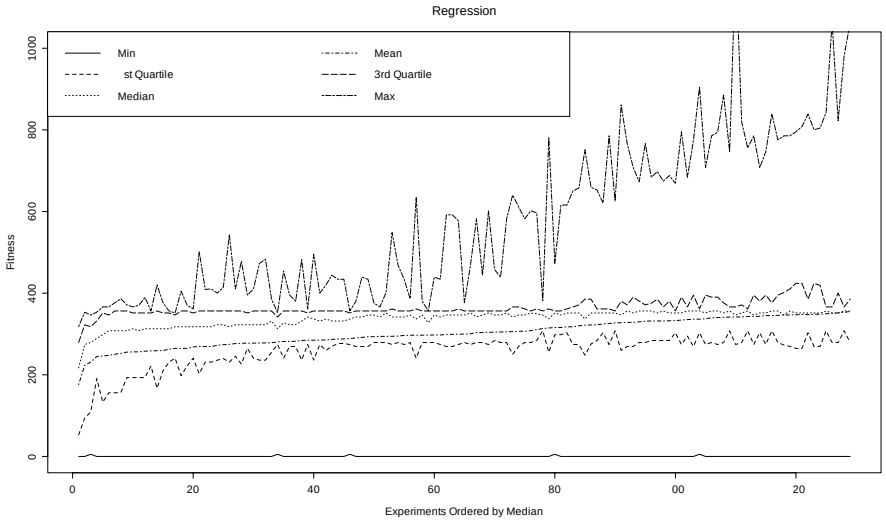


Fig. 4. Graphical representation of the statistics table of the GP-Hash problem when using the mean for ordering the shapes

problem[3]. In the table, the shapes on the top rows have a lower first quartile than those in the tail. In Figure 4, we can see a graphical representation of the whole table (Figure 5 shows similar representations of 3 other problems). All the statistics show a growing tendency when ordering by the mean. That means:

[3] Due to space limitations we cannot show all the tables here. We choose the Regression problem because it exemplifies very well the explanation. We have the same problem in Figure 5: we can only show 3 graphical representations in order to keep the figures legible.

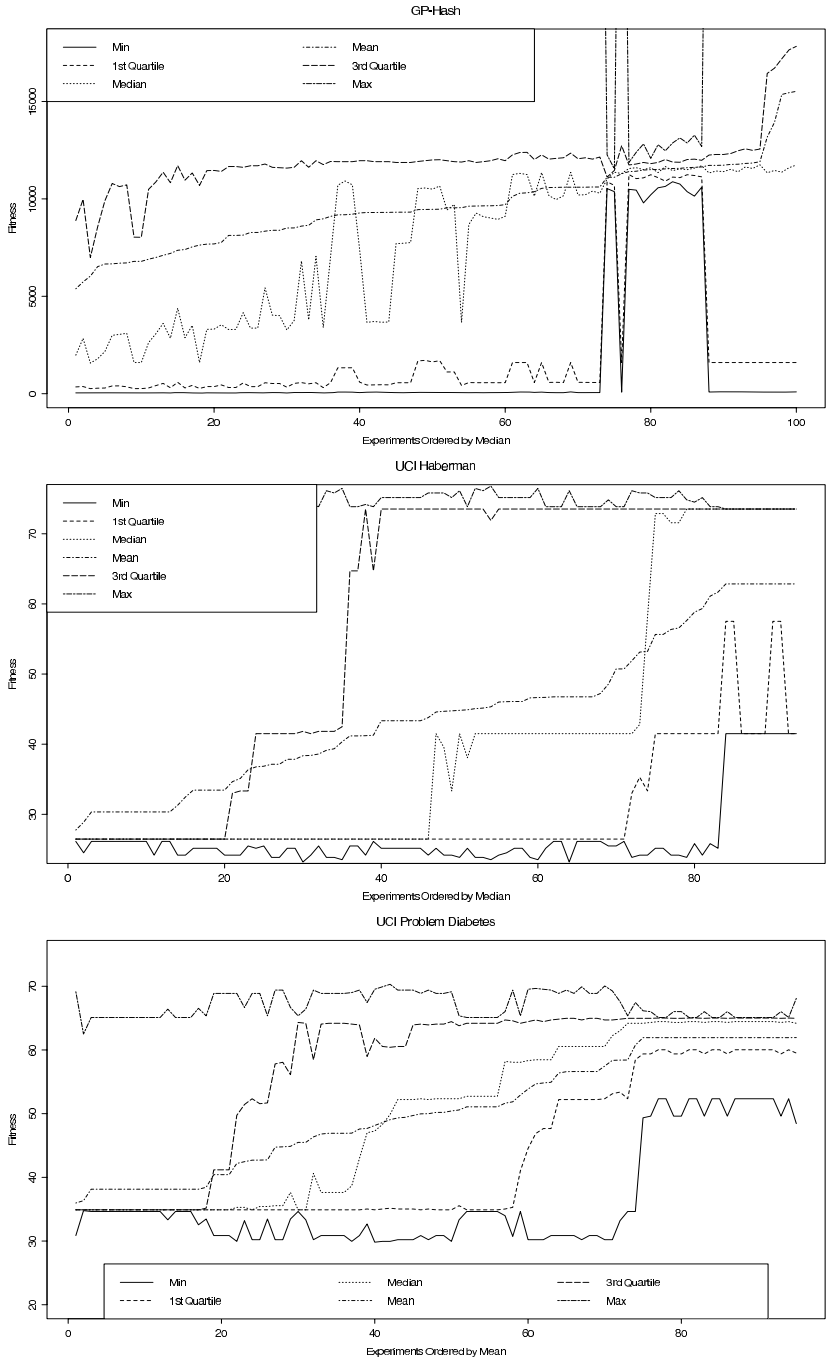


Fig. 5. Graphical representation of the statistics tables of other 3 problems (GP-Hash, Haberman, and Diabetes). Always ordering by mean.

the closer we get to the minimum mean value, the better the tree shapes are (according to our criteria). Of course, the tendency is not perfect: the growths of some statistics are irregular, but they are constant considering the big picture. So the mean could be a good estimator.

4 Efficiency and Computability

The problem now is how to estimate the mean fitness of a tree shape using an affordable amount of CPU power. In our experiments we have the complete distributions, but to obtain them we needed a huge computational effort. If we want to find a practical application of the explicit evolution of shapes, we need to be sure that we can make reliable and efficient estimations out of small random samples.

In a hypothetic implementation of our method, we have a population of tree shapes and we need to obtain the mean fitness of each one. We can extract a random sample from each shape, and estimate the mean. But we need to decide the size of the sample, trying to achieve a balance between estimation accuracy and CPU consumption. With our complete fitness distributions, we can use Chebyshev Inequality:

$$P\left[\frac{|X-\mu|}{\sigma} < k\right] > 1 - 1/k^2 \tag{1}$$

If we fix a value for k, then we can calculate which is the minimum size n which guarantees that the error between our estimation and the population mean is no greater than $k\sigma$ with a probability $p \geq 1 - 1/k^2$. But the problem is that if we do not have the complete distribution (and we will not in a real situation), we need to estimate the values of μ and σ, which means that we need to use a pilot sample, which means more CPU time. Instead, we will try to experimentally find a sample size that could be at least approximately correct for every problem. We tried with four different sample sizes: $n = 100$, $n = 30$, $n = 15$ and $n = 10$. For every tree shape of every problem we take 100 samples of size n. We perform a Wilcoxon/Mann-Whitney contrast (95% confidence level), comparing the mean of each sample with the mean of the complete distribution. If p value of a sample is greater than 0.05, then we have no reasons to say that there are significant differences between the average of the sample and the real mean. We call that an *accepted sample*. Using the information of the 100 tests, we can compute a proportion of *accepted samples* for a given shape. Finally, combining the proportions of each tree shape, we obtain the average proportion of *accepted samples* of size n for a given problem. We show this results on Table 3. We can see that even for the smallest sample size $n = 10$, we can accurately estimate the mean fitness of a distribution around 95% of the times (94.97% in the Regression problem, above 95% in all the other problems). This is an important result, because given that most of the fitness distributions we studied have thousands or millions of matches, it is impressive to be able to make precise

Table 3. Proportion of *accepted samples* for each sample size (100, 30, 15, 10) and for each problem

	$n = 100$	$n = 30$	$n = 15$	$n = 10$
Regression	95.48%	95.34%	95.06%	94.97%
6-Multiplexer	96.69%	95.79%	95.74%	95.13%
GP-Hash	97.77%	96.87%	96.56%	96.28%
Diabetes	96.74%	96.28%	96.02%	95.76%
SpecHeart	95.73%	95.72%	95.71%	95.38%
Haberman	97.04%	96.47%	95.72%	95.62%

estimations using only 10 samples. We believe that this is possible because of the intense grouping around the mode that we observed in the fitness distributions.

5 Conclusions

One-Point Crossover is an alternative to the Standard Crossover operator. It makes the GP population to converge to a common tree shape. The criterion that One-Point Crossover uses to guide the convergence is the fitness of the matches of that tree shape in the population (and also the fitness of individuals that contains the building blocks of that shape). This could be understand as an implicit evolution of tree shapes.

Our hypothesis is that an explicit evolution of tree shapes could improve the results obtained by One-Point Crossover. The goal of this work was to find an alternative way to explicitly measure how good a tree shape is. Our experiments show that just by sampling the distributions of fitness and extracting the mean value, we can obtain a reasonably good criterion to prefer some tree shapes rather than others. This sampling, to be optimum, only needs a small number of samples to give reliable estimations of the mean.

Obviously, the final justification of this work is to implement this explicit evaluation method and see if it improves the GP search capabilities as we expect. The idea is to make One-Point Crossover work in two different phases: in the first phase, it performs a search on partitions of the search space (explicit evolution of tree shapes), evaluating how promising the partitions are. This evaluation is made considering the mean, and relying in the experimental results obtained in this work: tree shapes with a good mean fitness, have a high concentration of better-than-average fitness solutions. Then, in the second phase, we already have a winner shape, and we can focus on the region delimited by it. This new method is analogous to One-Point Crossover, the only difference being that we expect the explicit evolution of shapes to make the first phase more controlled and rigorous. Our long term objective is to check if that really translates into an improvement of the search power of GP.

Acknowledgements

This work has been funded by the Spanish Ministry of Education and Science and FEDER under contract TIN2005-08818-C04 (the OPLINK project) and by

Comunidad de Madrid under contract 2008/00035/001 (Técnicas de Aprendizaje Automático Aplicadas al Interfaz Cerebro-Ordenador).

References

1. Koza, J.R., Andre, D., Bennett III, F.H., Keane, M.: Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufmann, San Francisco (1999)
2. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Dordrecht (2003)
3. Nordin, P., Banzhaf, W.: Complexity compression and evolution. In: Eshelman, L. (ed.) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 1995), pp. 310–317. Morgan Kaufmann, San Francisco (1995)
4. Langdon, W.B., Soule, T., Poli, R., Foster, J.A.: The evolution of size and shape. In: Spector, L., Langdon, W.B., O'Reilly, U.-M., Angeline, P.J. (eds.) Advances in Genetic Programming 3, ch. 8, pp. 163–190. MIT Press, Cambridge (1999)
5. McPhee, N.F., Miller, J.D.: Accurate replication in genetic programming. In: Eshelman, L. (ed.) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 1995), Pittsburgh, PA, USA, pp. 303–309. Morgan Kaufmann, San Francisco (1995)
6. Blickle, T., Thiele, L.: Genetic programming and redundancy. In: Hopf, J. (ed.) Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI 1994, Saarbrücken), pp. 33–38. Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany. Max-Planck-Institut für Informatik, MPI-I-94-241 (1994)
7. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
8. Soule, T.: Code Growth in Genetic Programming. Ph.D thesis, University of Idaho, Moscow, Idaho, USA (May 15, 1998)
9. Bhattacharya, M., Nath, B.: Genetic programming: A review of some concerns. In: Alexandrov, V.N., Dongarra, J., Juliano, B.A., Renner, R.S., Tan, C.J.K. (eds.) ICCS-ComputSci 2001. LNCS, vol. 2074, pp. 1031–1040. Springer, Heidelberg (2001)
10. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Heidelberg (2002)
11. Poli, R., Langdon, W.B.: On the search properties of different crossover operators in genetic programming. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, pp. 293–301. Morgan Kaufmann, San Francisco (1998)
12. Luke, S., Spector, L.: A comparison of crossover and mutation in genetic programming. In: Koza, J.R., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M., Iba, H., Riolo, R.L. (eds.) Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA, pp. 240–248. Morgan Kaufmann, San Francisco (1997)
13. Angeline, P.J.: Subtree crossover: Building block engine or macromutation? In: Koza, J.R., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M., Iba, H., Riolo, R.L. (eds.) Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA, pp. 9–17. Morgan Kaufmann, San Francisco (1997)

14. Estebanez, C., Hernandez-Castro, J.C., Ribagorda, A., Isasi, P.: Finding state-of-the-art non-cryptographic hashes with genetic programming. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervos, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 818–827. Springer, Heidelberg (2006)
15. Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S.: Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In: Greenes, R.A. (ed.) Proceedings of the Symposium on Computer Applications in Medical Care, pp. 261–265. IEEE Computer Society Press, Los Alamitos (1988)
16. Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M.R., Goodenday, L.S.: Knowledge discovery approach to automated cardiac SPECT diagnosis. Artificial Intelligence in Medicine 23(2), 149–169 (2001)
17. Haberman, S.J.: Generalized residuals for log-linear models. In: Proceedings of the 9th International Biometrics Conference, Boston, pp. 104–122 (1976)